On Algorithmic Rate-Coded AER Generation

Alejandro Linares-Barranco, Gabriel Jimenez-Moreno, Bernabé Linares-Barranco, and Antón Civit-Balcells

***Abstract*—This paper addresses the problem of converting a conventional video stream based on sequences of frames into the spike event-based representation known as the address-event-representation (AER). In this paper we concentrate on rate-coded AER. The problem is addressed as an algorithmic problem, in which different methods are proposed, implemented and tested through software algorithms. The proposed algorithms are comparatively evaluated according to different criteria. Emphasis is put on the potential of such algorithms for a) doing the frame-based to event-based representation in real time, and b) that the resulting event streams ressemble as much as possible those generated naturally by rate-coded address-event VLSI chips, such as silicon AER retinae. It is found that simple and straightforward algorithms tend to have high potential for real time but produce event distributions that differ considerably from those obtained in AER VLSI chips. On the other hand, sophisticated algorithms that yield better event distributions are not efficient for real time operations. The methods based on linear-feedback-shift-register (LFSR) pseudorandom number generation is a good compromise, which is feasible for real time and yield reasonably well distributed events in time. Our software experiments, on a 1.6-GHz Pentium IV, show that at 50% AER bus load the proposed algorithms require between 0.011 and 1.14 ms per 8 bit-pixel per frame. One of the proposed LFSR methods is implemented in real time hardware using a prototyping board that includes a VirtexE 300 FPGA. The demonstration hardware is capable of transforming frames of 64×64 pixels of 8-bit depth at a frame rate of 25 frames per second, producing spike events at a peak rate of 10^7 events per second.**

I. Introduction

PRESENT day computer and electronic technology allows for very efficient and powerful intelligent artefacts. However, there are a number of applications in which engineers would desire a more profound understanding of how biological living brains perform cognitive human-like tasks. Biological brains are based on clumsy and slow elements called *neurons*, which work during a limited lifetime only. However, living brains use them with massive parallelism, under clever hierarchical structures which adapt, self-correct, circumvent component imperfections, and achieve outstanding computing performance for what we know as human-like cognitive tasks, such as driving cars, talking to people, playing football or tennis, adapting to new environments, and so on. Particularly, human vision achieves outstanding performance in tasks such as scene segmentation and analysis, shape recognition and reconstruction from occluded and distorted patterns, rapid identification of unknown shapes, etc. In conclusion, living brains overwhelmingly outperform their manmade artificial counterparts. An engineering understanding of the computational principles of biology would permit the construction of a new generation of artificial systems for human-like cognitive tasks with potential in applications such as vehicle navigation, prosthetics, pattern recognition, robotics, intelligent surveillance, and so on.

Primate brains are structured in layers of neurons, in which the neurons in a layer connect to a very large number $(\sim 10^4)$ of neurons in the following layer [1]. Many times the connectivity includes paths between nonconsecutive layers, and even feedback connections are present. Artificial bioinspired software models based on such connectivity models have overwhelmed the specialized literature presenting many ways of performing bioinspired processing systems that outperform more conventionally engineered machines [2]–[6]. Since these models are software based, they operate at extremely low speeds, because of the massive connectivity they emulate. For real-time solutions, direct hardware implementations are required. However, hardware engineers face a very strong barrier when trying to mimic the bioinspired hierarchically layered structure: The massive connectivity. In present day, state-of-the-art very large-scale integrated (VLSI) circuit technologies it is plausible to fabricate on a single chip many thousands (even millions) of artificial neurons or simple processing cells. However, it is not viable to connect physically each of them to even a few hundreds of other neurons. The problem is greater for multichip multilayer hierarchically structured bioinspired systems. address-event-representation (AER) is an incipient bioinspired spike-based technique capable of providing a hardware solution to the interchip massive connectivity problem.

AER-based interchip communication was originally proposed by Mahowald and Sivilotti [7]–[9] to reproduce the state of a two-dimensional (2-D) array of neurons from one emitter chip onto another receiver chip, continuously and in real time. A growing community of researchers is using this scheme for bioinspired vision [10]–[16] and audition [17] systems. The scheme has been evolving in efficiency and processing power [18]–[32]. AER technology has been exploited also to implement real time convolution operations, either of fixed [12], [31] or programmable kernel shape [15], [16]. In the last few years AER has been an important mainstream line at the annual National Science Foundation (NSF) funded Telluride Neuromorphic Engineering Workshop series [33].

Let us explain briefly why AER seems so promising to this research community. Biological neurons in a layer communicate with neurons in other layers by sending trains of spikes.

Manuscript received June 16, 2004; revised October 21, 2005. This work was supported in part by EU Grant IST-2001-34124 (CAVIAR), and by Spanish Grants TIC-2000-0406-P4 (VICTOR) and TIC-2003-08164-C03-01 (SAMANTA).

A. Linares-Barranco, G. Jimenez-Moreno, and A. Civit-Balcells are with the Arquitectura y Tecnología de Computadores, ETSI Informática, 41012 Sevilla, Spain (e-mail: alinares@atc.us.es).

B. Linares-Barranco is with the Instituto Microelectronica Sevilla (IMSE) National Microelectronics Center, CNM-CSIC Ed. CICA, 41012 Sevilla, Spain (e-mail: Bernabe.Linares@imse.cnm.es).

Digital Object Identifier 10.1109/TNN.2006.872253

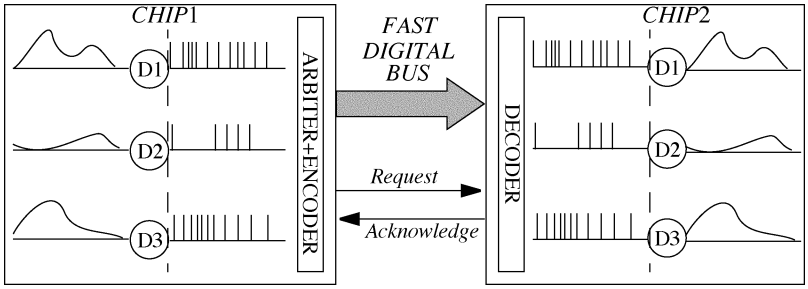


Fig. 1. AER interchip point-to-point communication scheme.

The intervals between spikes are in the order of milliseconds, or more. In artificial VLSI hardware systems it is possible to have the neurons of one layer on a single chip, but it is completely impossible to connect physically each neuron in a layer on one chip to other neurons on another layer at another chip. If each chip has for example 128×128 neurons and we need to connect each neuron to a projective field of 10×10 neurons on another chip, we would require more than 1.5×10^6 connections. At present the most advanced chip packages have a maximum of around 200–2000 pins. However, current VLSI technologies can easily handle transition times for interchip communications in the order of nanoseconds and less [34]. This is more than a million times faster than biological neurons. AER systems exploit this feature by: 1) Time-multiplexing the high degree of connectivity using high speed interchip digital buses with a reduced number of pins; and 2) by ensuring that only information carrying neurons consume communication bandwidth.

AER allows for different signal coding schemes, as will be explained later. The most direct and original one is called *rate-coding* (or *frequency-coding*). Fig. 1 illustrates the idea behind the rate-coded AER basics. An emitter chip contains an array of cells in which each pixel shows a continuously varying time dependent state which changes with a slow time constant, in the order of milliseconds. This would be the case, for example, of a camera or retina chip where each pixel includes a photosensor. The current through the photosensor would change continuously in time with a time constant in the order of milliseconds.[1] In Fig. 1, each pixel includes a local circuit that generates spikes, also called *events*. Each spike is of a very short duration, in the order of nanoseconds. The intervals between spikes are much larger, in the order of milliseconds or more. The density of spikes is proportional to the state or intensity of the pixel. This can be achieved by using a local oscillator which is a voltage controlled oscillator (VCO) whose frequency depends on pixel intensity. Another alternative could be to use integrate-and-fire neurons [38], [39]. Every time a pixel generates an event, it tries to communicate with the external interchip high speed bus. In the receiver chip, every time an event is received, the corresponding address is decoded and a spike is sent to the pixel located at that address. Thus the pixels with the same address in the receiver and emitter chips "see" the same sequence of spikes. The receiver pixel includes some type of integration mechanism, so that the original continuous time-varying state or intensity is recovered. This is *rate-* or *frequency-coded* AER, because pixel activity is transformed into pixel event frequency. Note that this coding scheme may be highly inefficient for convential image transmission: Monochrome VGA resolution[2] yields a peak rate of $(480{*}640 \text{ pixels/frame}) * (256 \text{ spikes/pixel}) * (25 \text{ frames/s}) * (19\text{bit/spike}) = 37$ Gbit/s. However, most of the AER hardware systems reported so far in literature use this *rate-coded* AER [7], [8], [10]–[12], [14]–[19], [23], [27], [29], [48], [50], [51]. This is because AER systems do not usually transmit raw images, but already preprocessed images, such as edges or contrast [51] (in which 20 gray levels are satisfactory, and only a small percentage of all pixels—between 1–10%—will present appreciable contrast). This will reduce the previous full VGA peak rate by two–three orders of magnitude. Also, present day AER hardware uses image resolutions between $64 * 64$ and $128 * 128$ pixels at the most, thus adding another one–two order reduction in the peak rate. Thus, present day rate-coded AER hardware requires around 200 Kevents/s (2.9 Mbit/s). In general, AER is useful for multistage processing systems, in which as events are generated at the front end they travel and are processed down the whole chain (without waiting to finish processing each frame). Also, in multistage systems, information is reduced after each stage, thus reducing the event traffic. AER is definitely not advantageous for simple image transmission and restoration systems.

New findings in neuroscience have revealed that besides the historically postulated frequency coding scheme of spikes, brains exhibit other coding schemes which do not require the integration of spike trains, a relatively slow and inefficient process. Researchers have discovered that brains use space coded spike patterns to transmit information [35]–[38]. Consequently, AER can be used to generate almost simultaneous trains of space-time correlated events that code information. This is, for example, the case of motion retinae, in which a moving profile elicits a train of simultaneous events at the coordinates of the profile. Other recent findings reveal that brains use intensity to *spike-time* coding, in which *spike-time* is the delay between a global reset time and spike appearance [38]–[40]. Pixels of high intensity would appear almost instantaneously, while low intensity pixels would appear later. The global reset time could be established by the appearance of a new image, for example. This coding scheme would also produce almost instantaneous and highly space-time correlated

[1]For example, in commercial video cameras each pixel is sampled between 25–30 times per second, which is a sampling period between 33–40 ms (this rate is called *frame rate*). According to the sampling theorem, the pixel signal bandwidth should be less than half the sampling rate. Therefore, less than 12.5–15 Hz.

[2]480×640 pixel frames, at 25 frames per second, with 8 bits per pixel.

spike events corresponding to profiles of the most intense image pixels.

In this paper, we will concentrate on rate-coded AER, which is still the most widely reported for AER hardware systems [7], [8], [10]–[12], [14]–[19], [23], [27], [29], [48], [50], [51]. Research for nonrate-coded AER algorithmic generation is presently under development [40]–[43]. Nonetheless, the rate-coded-based algorithms analyzed in this paper do not lose the space-time correlation property of events belonging to the same simultaneous profile, as will be illustrated at the end of the paper.

A very interesting and powerful feature of AER communication is its potential for adding computations on the fly. Since pixel addresses are physically present on the interchip bus, extra digital address processing is possible. Simple combinational circuits can be inserted in the data path to perform shifting of the address space. Inserting (EE) PROMs with appropriate lookup tables would allow rotation operations or generic remappings. More complicated circuits can also be used such as microcontrollers, which, from a single address, can generate sequentially a "bubble" of addresses around it, also called a "projection field" [29]. This freezes the interchip communications while the projection field is being generated. Consequently, this approach introduces a significant delay which grows quadratically with the projection field radius. Interesting and powerful approaches for performing AER-based convolution operations have also been proposed [12], [15], [16] by developing special receiver chips which would not freeze the interchip high-speed bus.

AER not only offers the possibility of great and powerful computations while events travel between chips but also an easy way of expanding system size by assembling multichip hierarchical structures. If a neural layer is required with more neurons than those on a single chip, it is possible to arrange chips in a matrix fashion in order to assemble a layer of the required size. Also, multiple layers are directly assembled by connecting chips through the AER high speed interchip buses. Fig. 1 shows a typical AER point-to-point communication [23]. However, this scheme could be easily expanded to multireceiver [26] and multisender situations [24]. Consequently, AER can easily evolve into a technology that offers the possibility of assembling arbitrarily complex hierarchically structured multilayered neural systems. Researchers and system developers would be able to assemble systems by using simple building blocks such as AER retina chips, convolution transceivers, competition transceivers, ... and interchip AER buses. When scaling up AER-based systems, care should be taken to guarantee that each AER bus does not require an event traffic sustain rate greater than the physical limit it allows. In practice, this is achieved by scaling down the maximum event frequency of the pixels.

However, for proper development, usage and exploitation of this AER technology a key element is still missing. This key element is a proper computer interface that would allow us to "probe" the interchip AER buses. By "probing" we mean either: 1) To visualize the 2-D images that travel from chip to chip coded as address events; or 2) to inject artificial images stored or generated by the computer into an AER interchip bus. This way, developers would be able to work on later stages of a complicated structure without having all the preceding layers ready. For example, when developing our AER convolution chips [16], the FBR-AER computer interface proved to be a crucial piece for proper testing, characterization, tuning and debugging. Such interfaces require transformations in real time between the asynchronous AER spikes and the conventional FBR (frame-based representation) video sequences of conventional video systems.

However, transforming the video stream of a conventional frame-based representation (FBR) sequence to an asynchronous event-based representation in real time is not trivial at all. On the other hand, going from asynchronous AER to synchronous FBR video is more or less straightforward: If T_{frame} is the duration of a single frame, a 2-D video frame memory is reset at every time $t = nT_{\text{frame}} (n \in [0, \infty[)$, then for each event address (x, y) the memory position for this address is incremented by 1; at $t = (n+1)T_{\text{frame}}$ the content of the 2-D memory is transferred to the computer screen and reset again. This is more or less how state-of-the-art AER hardware engineers visualize their AER systems outputs on computers [14]. On some occasions integrating AER receivers are built which are read out periodically as frames (FBR) [19]. On the other hand, the transformation from a conventional FBR video sequence coming from a computer (or any other conventional video device) to an asynchronous AER is much more sophisticated, especially if done in real time. Such a transformation is the purpose of the present paper. We have studied several algorithms able to perform such a task and have compared them according to different criteria. At some point the computer will interface to the physical AER bus. Consequently, if the algorithm is fast enough, the transformation from FBR to AER can be performed in software and the interfacing hardware would simply transmit the address events from inside the computer to the AER chips buses. However, as image size and event activity increases, the computational load increases quadratically, and the software transformation is impractical at some point. Under this situation, it would be desirable to implement the algorithms directly in hardware in order to speed up the transformations. Our objective in this paper is to show algorithms and techniques which can generate rate-coded AER activity in real-time using the standard PCI bus of a conventional computer. Then we will show how we have implemented one of these algorithms in real time FPGA-based hardware.

The paper is structured as follows. Section II describes several algorithms for FBR-to-AER transformations. In Section III these algorithms are evaluated comparatively according to different criteria. Finally, Section IV describes a hardware FPGA-based implementation of one of the proposed algorithms.

II. Algorithms for Rate-Coded AER Generation

One can think of many software algorithms to transform a 2-D bitmap image (stored in a computer's memory) into an AER stream of pixel addresses. Generally, the frequency of appearance of the address of a pixel should depend on the intensity of the pixel. If event frequency is much higher than the pixel signal bandwidth, then precise positioning of the events in time is not critical. Events can be time shifted up to a certain degree because AER receivers reconstruct signals by integrating (averaging) over time. If, for example, events are randomly shifted in

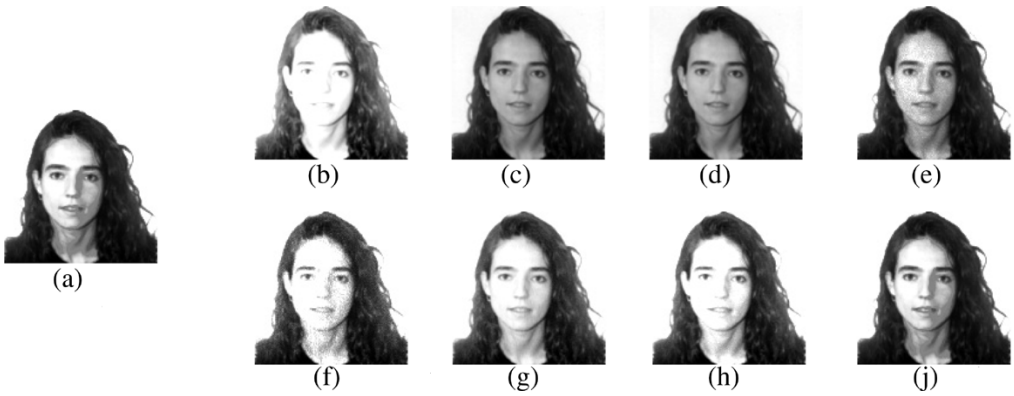


Fig. 2. Illustration of reconstructed AER images. On the left, an original 128 × 128 image with 256 gray levels, which has been transformed (by software) into AER streams using different algorithms. The other eight figures correspond to reconstructing the images by averaging (for each pixel) the interevent times over a complete frame period, and modulating the resulting value by an event distribution error term, as explained in the text. (a) Original image. (b) Scan. (c) Uniform-BF. (d) Uniform-F. (e) Uniform-WTA. (f) Random. (g) Random-SQ. (h) Random-HW. (j) Exhaustive.

time with respect to their ideal exact position, then after image reconstruction (integration) the image would appear with some added noise.

The algorithms should perform a transformation from pixel activity p to pixel event frequency f_p. Let us assume pixel activity is constrained to a normalized unity interval $p \in [0,1]$. As explained in the Appendix, we will assume some (optional) constraints throughout the paper, yields

$$f_p = p\frac{K}{T_{\text{frame}}} \tag{1}$$

where $K = n_{\text{res}}$ is linear pixel resolution.[3] Whatever algorithm is used, the idea is to generate for each video frame a vector of addresses that will be sent to an AER bus. Let us call this vector the "*frame vector*." The *frame vector* has a fixed number of time slots to be filled with event addresses. The number of time slots depends on the time assigned to a frame (for example $T_{\text{frame}} = 40\,\text{ms}$) and the time required to transmit a single event (for example $T_{\text{event}} = 10\,\text{ns}$). The total number of time slots in the *frame vector* is then $n_{\text{slots}} = T_{\text{frame}}/T_{\text{event}}$. If we have an image of $N \times M$ pixels and each pixel has a gray-scale resolution of n_{res}, then the maximum possible number of events one pixel can put on the "*frame vector*" is $K = n_{\text{res}}$ (for $p = 1$). In the worst case, the *frame vector* would be filled with $N \times M \times K$ addresses. Note that, in principle, one would like this number to be less than or equal to n_{slots}, although this restriction is not critical since the situation of having all pixels at the maximum rate is not realistic. Depending on the total intensity of the image there will be more or less empty slots in the *frame vector*. Each algorithm would implement a particular way of distributing these address events, and would require a certain computation time. Note that this frame vector is a very memory hungry buffer (for 256 × 256 pixel images, with $T_{\text{event}} = 10\,\text{ns}$ and $T_{\text{frame}} = 40\,\text{ms}$, it requires 8 MBytes). For real-time hardware, the most attractive algorithms are those that sweep the frame vector once, slot after slot. This way it is not necessary to build this large memory frame vector buffer physically.

Next, we propose some algorithms capable of performing the transformation FBR-to-AER [44], [45]. Then, in the following section we will explain criteria to evaluate and compare the different algorithms.

[3]For example, for 8-bit pixel resolution $n_{\text{res}} = 2^8 = 256$.

A. Scan Algorithm

Assume an image (or frame) of an FBR system is stored during a time T_{frame} in a temporary (RAM) memory. Each memory position contains an integer with values between 0 and $K - 1$. This algorithm scans the memory pixel by pixel many times, while a pointer sweeps the *frame vector* slot by slot. If the pixel value is nonzero, its address is put on the *frame vector* at the current *frame vector* pointer position. The pixel value is decremented by one, and the *frame vector* pointer is incremented by one. If the pixel value is zero, a blank slot is left in the *frame vector*. This method is computationally simple and consequently relatively fast. However, the resulting event distribution is very different from the one an AER emitter chip would produce. Particularly, events would be much more concentrated at the beginning of the frame vector, getting more and more sparse along it, and eventually the frame vector could be completely empty in the last positions.

As an illustration, Fig. 2 shows reconstructions of an image that has been transformed into different AER streams for different AER generation algorithms. The original image in Fig. 2(a) was used to generate an AER stream for the time of one frame (40 ms). Then, for each pixel (i.e., for events of equal address in the AER stream), the interevent times were averaged over the whole *frame vector*, and the resulting value was taken as the reconstructed value for that pixel. This value is exactly n_{evf} for that pixel,[4] as in the ideal AER distribution. In order to illustrate the difference between an ideal AER distribution and the one generated by the algorithm, we multiply the reconstructed value by an error term[5] that indicates how much the interevent intervals differ between the ideal and reconstructed distributions for a frame. In Section III we will return to these issues with precise mathematical considerations. Fig. 2(b) shows the resulting values of the pixels for the AER stream generated by the *Scan Algorithm*.

[4]Except for algorithms where events are deleted after a collision.

[5]The error term is $1+\text{error}$, where error is the average (over a frame) relative norm-1 distance between ideal and real interevent times.

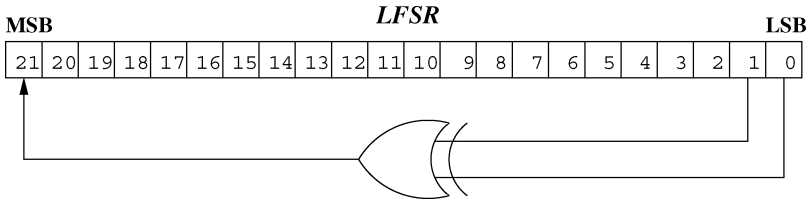


Fig. 3. Random-Hardware LFSR uses a full 22-bit register.

B. Uniform Algorithm

The objective of this algorithm, as opposed to the previous one, is to distribute equidistantly the events of each pixel along the *frame vector*. A frame is scanned pixel by pixel only once. The intention is now for each pixel to distribute its events at equal distances along the *frame vector*. As the *frame vector* is being filled, the algorithm may want to place addresses in slots that are already occupied. This situation is called a *collision*, and one of the two events needs to be reallocated. In this case, we propose three solutions.

The *Back-Forward* (Uniform-BF algorithm) solution will put the event in the nearest empty slot of the *frame vector*. Fig. 2(c) shows the reconstructed error-modulated AER stream generated from the image in Fig. 2(a) using this algorithm.

The *Forward* (Uniform-F algorithm) solution will put the event in the following empty slot of the *frame vector*. Fig. 2(d) shows the reconstructed error-modulated AER stream generated from the image in Fig. 2(a) using this algorithm.

And the *Winner-Take-All* (Uniform-WTA algorithm) solution will put the event of lowest intensity in the collision position of the *frame vector*. The event of the other pixel of higher intensity will be ignored. Theoretically, pixels with higher intensity appear more frequently on the bus. Therefore, ignoring one of their events will produce a smaller event distribution error along the frame vector than for less active pixels. Fig. 2(e) shows the reconstructed error-modulated AER stream generated from the image in Fig. 2(a) using this algorithm.

Uniform-BF, *Uniform-F* and *Uniform-WTA* algorithms will introduce more distribution errors (reallocation of events) at the end of the process than at the beginning. The execution time grows considerably because the collisions consume an important amount of time to be resolved.

C. Random Algorithm

This algorithm exploits the traditional method of random number generation based on linear feedback shift registers (LFSR) [46]. LFSR-based random number generation provides a very long sequence of random numbers repeated periodically. If the shift register is of n-bit size, then the period will be $2^n - 1$ (number zero is excluded). All possible numbers of n-bit size will be generated (except zero) but none of them will be repeated during the same period. Consequently, if we make the *frame vector* of size $n_{\text{slots}} = 2^n$, we can use an LFSR of n-bit size and generate a random sequence for the *frame vector* positions into which events may be allocated. For example, if each frame is of size $N \times M = 128 \times 128$ and each pixel generates between 0 and $K - 1 = 255$ events per frame n_{evf}, then the maximum number of slots required for the *frame vector* would be

$$n_{\text{slots}} = K \times N \times M = 256 \times 128 \times 128 = 2^8 \times 2^7 \times 2^7 = 2^{22} \tag{2}$$

and an LFSR of 22-bit size could be used. For example, Fig. 3 shows a 22-bit LFSR that would generate all 2^{22} numbers (except '0') in a "random" order. The theory of LFSR explains how to implement the feedback logic for pseudorandomness to occur, and what is the minimum bits that need to be fed back, depending on register size [46].

An LFSR random algorithm for FBR-AER transformation would operate as follows. A frame is stored into the temporary memory. The frame is scanned pixel by pixel only once. For each pixel, the LFSR is called as many times as the value of n_{evf}, and the pixel address is placed on the *frame vector* at the positions provided by the LFSR.[6] The events of the same pixel will not be equidistant along the *frame vector*, but will be distributed randomly over its complete range. Unfortunately, it is possible to allocate for the same pixel two events very close in the *frame vector*. This is not desirable and can be avoided in several ways. For example, instead of using an LFSR of size n for the *frame vector* pointer, one can use one of size $n - b$ for the lower bits of the *frame vector* pointer and a b-bit counter for the most significant bits. The algorithm would operate as follows. The frame memory is scanned once. Then, for every 2^b pointer values, the $n-b$ bit LFSR is called once and the counter is called 2^b times. This will produce 2^b equally spaced pointers along the *frame vector* whose initial pointer is randomly placed along the first 2^b positions.

Fig. 4 shows, for example, an LFSR structure with a 2-bit counter for a 128×128 frame with 256 gray levels $(n-22\ \text{bits})$. This would divide the *frame vector* into four sections. A random pointer would be generated for the first section and three more equally distant pointers along the whole *frame vector* would follow. If the number of sections is selected to be equal to the maximum possible value of $n_{\text{evf}}|_{\max} = K$, then events of the same pixel address would never fall within the same section for the same frame.

Fig. 2(f) shows the reconstructed error-modulated AER stream generated from the image in Fig. 2(a) using this algorithm with a 2-bit counter.

D. Random-Square Algorithm

For the *Random* algorithm with a fixed size counter from 1 to the maximum n_{evf} (or maximum pixel gray level $K - 1$), the event distribution for high activity pixels is acceptable, but

[6]Since number zero cannot be generated, the algorithm can use this position for the first event and afterwards call the LFSR for the rest of events.

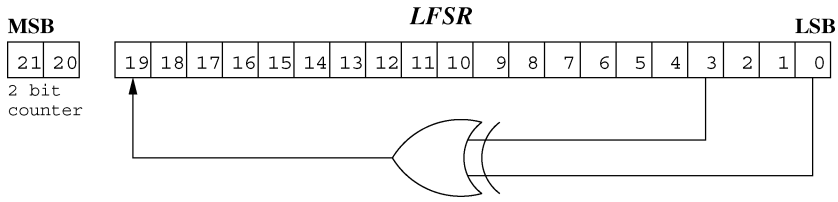


Fig. 4. Random method structure: LFSR with a 2-bit counter.

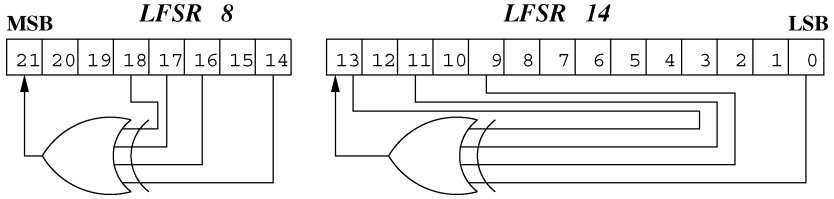


Fig. 5. Random-Square structure: LFSR-8 and LFSR-14.

poor for low level values (they will be always concentrated at 2^b distances in the *frame vector*). Substituting the counter by another LFSR, the distribution could be improved.

For an $N \times M = 128 \times 128 = 2^7 \times 2^7$ frame with maximum $n_{\rm evf}$ of $K-1 = 255 = 2^8-1$, an 8-bit $(\log_2(K))$ LFSR (LFSR-8) is used for selecting 255 slices of 128×128 positions, and another 14-bit $(\log_2(N) + \log_2(M))$ LFSR selects the position inside the slice. The image is scanned only once. For each pixel a 14-bit number is generated by the LFSR-14, and the LFSR-8 is called as many times as $n_{\rm evf}$. Fig. 5 shows the LFSRs used by this random-square method. Fig. 2(g) shows the reconstructed error-modulated AER stream generated from the image in Fig. 2(a) using this algorithm.

E. Random-Hardware Algorithm

The two previous LFSR-based methods are very attractive for a hardware implementation because of the simplicity and efficiency of the LFSR methods. However, in both cases the complete *frame vector* has to be generated and stored before starting the transmission. This frame vector could be extremely large for medium-to-large frame sizes and medium-to-high pixel resolutions. For example, for $N = M = 128$ and $K = 256$, the maximum number of events in the frame vector can be as large as $n_{\rm slots} = N \times M \times K = 2^{22} = 4.19 \times 10^6$, while each slot needs to store $7 + 7 = 14$ bits. Therefore, these solutions are only useful for hardware implementations that require reasonably small *frame vectors*. So far, the only method that does not require the physical presence of a *frame vector* if implemented in hardware, is the *Scan Algorithm*. This is because the frame vector pointer advances uniformly from start to end. For all other methods, the pointer goes back and forth. The algorithm described in what follows sweeps the frame vector pointer uniformly, but is of random nature.

This algorithm only requires memory to store the frame image to be transformed. It uses an LFSR of as many bits as necessary to generate $N \times M \times K$ numbers, as before. For example, if $N = M = 2^7 = 128$ and $K = 2^8 = 256$, then $7 + 7 + 8 = 22$ bits are needed. The 22-bit LFSR is called 2^{22} times, providing random numbers. For each number, a pixel is selected in the image using the $\log_2(N) + \log_2(M)$ less significant bits of the pseudorandom number. With the other $\log_2(K)$ bits, the algorithm decides if the event has to be sent or not. If the $\log_2(K)$ more significant bits represent a number larger than the value of the pixel $(n_{\rm evf})$, then an event is sent with the $\log_2(N) + \log_2(M)$ less significant bits of the pseudorandom number as the address. In the other case, the pseudorandom number is ignored and a pause equivalent to one event is generated. Consequently, the algorithm generates the pseudorandom numbers, and decides whether or not the resulting event is sent in real time. Therefore, no *frame vector* is needed. Fig. 3 shows the LFSR used for this method in the case $N = M = 128$ and $K = 256$. Fig. 2(h) shows the reconstructed error-modulated AER stream generated from the image in Fig. 2(a) using this algorithm.

F. Exhaustive Algorithm

This algorithm [44], [45] divides the *frame vector* into K equal slices. Each slice is of size $N \times M$, so that it assigns the same position for each pixel within all slices (so far, this is like in the *Random Square Algorithm*, or the *Random Algorithm* for $2^b = K$). This algorithm assigns for the first pixel the first position in the slice, for the second pixel the second position in the slice, and so on. Consequently, it uses the same counter to sweep pixels in the frame as well as to generate the positions within the slice. The peculiar thing about this algorithm is the way it selects the slices into which the $n_{\rm evf}$ events are put for each particular pixel. Each selected slice will have one event for the active pixel, and the selected slices will be such that they are as far apart as possible. For example, if $K = 8$ and $n_{\rm evf} = 2$, then the maximum spacing is 4. The algorithm will put one event always in the last slice $(k = 7)$ and in slice $k = 3$ $(k \in [0, 7])$. Thus, if $n_{\rm evf}$ is constant from frame to frame, the distance will always be four slices. If $K = 8$ but $n_{\rm evf} = 3$, the algorithm will select slices $k = 2$, 5, 7 which will give distances of 2, 2, and 1 slices. Mathematically, one needs to find out if $(K/n_{\rm evf})n_i$ (with $n_i = 1, \ldots n_{\rm enf}$) falls within the kth slot. More precisely, if

$$k < \frac{K}{n_{\rm evf}} n_i \leq k + 1. \tag{3}$$

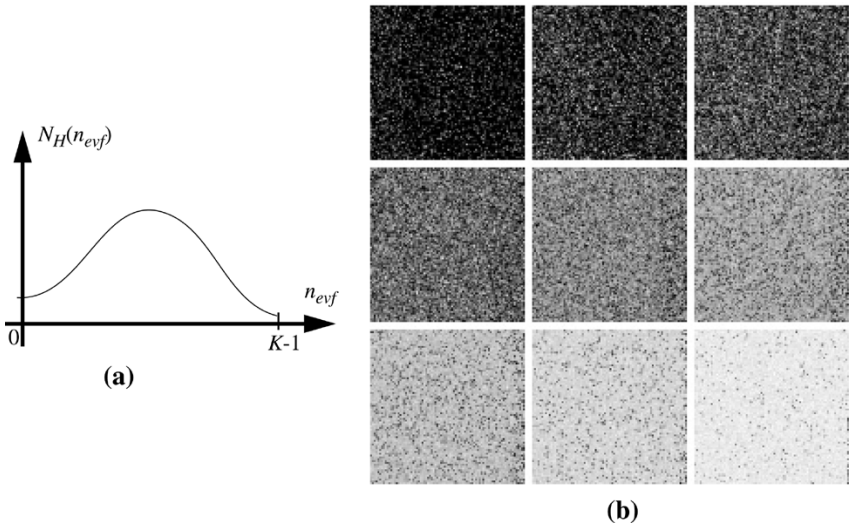


Fig. 6. TIS. (a) Histogram for random image generation. (b) Resulting images (10% load upper left, 90% load lower right.

This same condition can be expressed more elegantly as

$$(k \times n_{\rm evf}) \mathrm{modK} + \mathrm{n_{evf}} \geq \mathrm{K}. \tag{4}$$

The algorithm operates as follows. The frame is stored in local RAM memory, and is swept pixel by pixel K times. At the same time the *frame vector* pointer advances by one for each pixel, using a counter that counts up to $N \times M \times K$. The lower $\log_2(N) + \log_2(M)$ bits of the counter determine the position that corresponds to each pixel within the slices. This slice position is thus unique for each pixel. The higher $\log_2(K)$ bits of the counter tell us in which *frame vector* slice we are. For each slice $k \in [0, K-1]$ and the algorithm evaluates whether or not (4) is satisfied. If it is true, an event for this pixel is put on this slice. Otherwise, the slice is skipped for this pixel [44], [45]. Fig. 2(j) shows the reconstructed error-modulated AER stream generated from the image in Fig. 2(a) using this algorithm.

Note that this method does not require a physical implementation of the *frame vector*, because *frame vector* positions are swept one by one, as was done by the *Scan* and *Random-Hardware* algorithms.

III. Algorithm Evaluation and Comparison Results

In this section we compare the methods proposed above and estimate how the performance of the methods is affected by the traffic or load of events in the AER bus. To carry out this analysis we generated a set of images with random pixel values. With the assumptions and constraints that we have used in the previous section ($a = 1, \Delta n_{\rm evf} = 1$), an $N \times M$ frame with all pixels at maximum value $K - 1 = n_{\rm res} - 1$ would completely fill a "*frame vector*" of size $n_{\rm slots} = N \times M \times K$. In this case we say the frame has a 100% load on the AER bus. If a frame has an average pixel value of 50% then the *frame vector* is filled only up to 50%. The more load in the frame, the more difficult it will be for the FBR-AER algorithm to perform the transformation, yielding worse results. In order to test the previously outlined FBR-AER algorithms we generated nine different test images of different loads (10%, 20%, 30%, ... 80%, 90%). Let us call $q \in [0, 1]$ the load (normalized to unity) on the AER bus, and $N_H(n_{\rm evf})$ the number of pixels in a frame with value $n_{\rm evf}$. This function [see Fig. 6(a)] is precisely the histogram for the events of one frame. The area of the histogram $\sum N_H(n_{\rm evf})$ is equal to the total number of events n_{evT} and satisfies

$$n_{evT} = \sum_{n_{\rm evf}=1}^{K-1} N_H(n_{\rm evf}) = N \times M \times K \times q. \tag{5}$$

Using a Gaussian function with the constraint of (5) and imposing that either $N_H(1) = 1$ or $N_H(K-1) = 1$, a unique Gaussian function for $N_H(n_{\rm evf})$ is obtained. Then, for each $n_{\rm evf} \in [1, K-1]$, $N_H(n_{\rm evf})$, random addresses (x, y) of the frame image are generated into which pixel value $n_{\rm evf}$ is written. Thus, a random image of load q and a Gaussian histogram are generated. Fig. 6(b) shows the nine generated images ($N = M = 128, K = 256$). Let us call this set of nine images our Test Image Set (TIS). We will compare the FBR-to-AER algorithms according to five different criteria: Computational complexity, distribution error, distribution histogram, event clustering, and reconstruction error.

A. Computational Complexity

Let us compare the algorithms according to the time needed by the same computer to run them as software programs. This will compare the computational complexity of the algorithms. The algorithms were written in C. We are not interested in the absolute speed of the software, but rather in the relative difference between the execution times of the different algorithms. In order to do this, all algorithms were written in C doing the exact operations described above, without calling library routines. Each algorithm was executed for the nine images of our TIS, and execution times were normalized with respect to the maximum. The results are shown in Fig. 7(a). As may be seen, all uniform algorithms grow in computation time very quickly as the event load of the images increases. The *Scan Algorithm* is the fastest one since it hardly needs to execute any mathematical operations. Also, its execution time increases very slowly with the event load. The *exhaustive* algorithm is the next fastest one, since it requires some more computations, and its execution

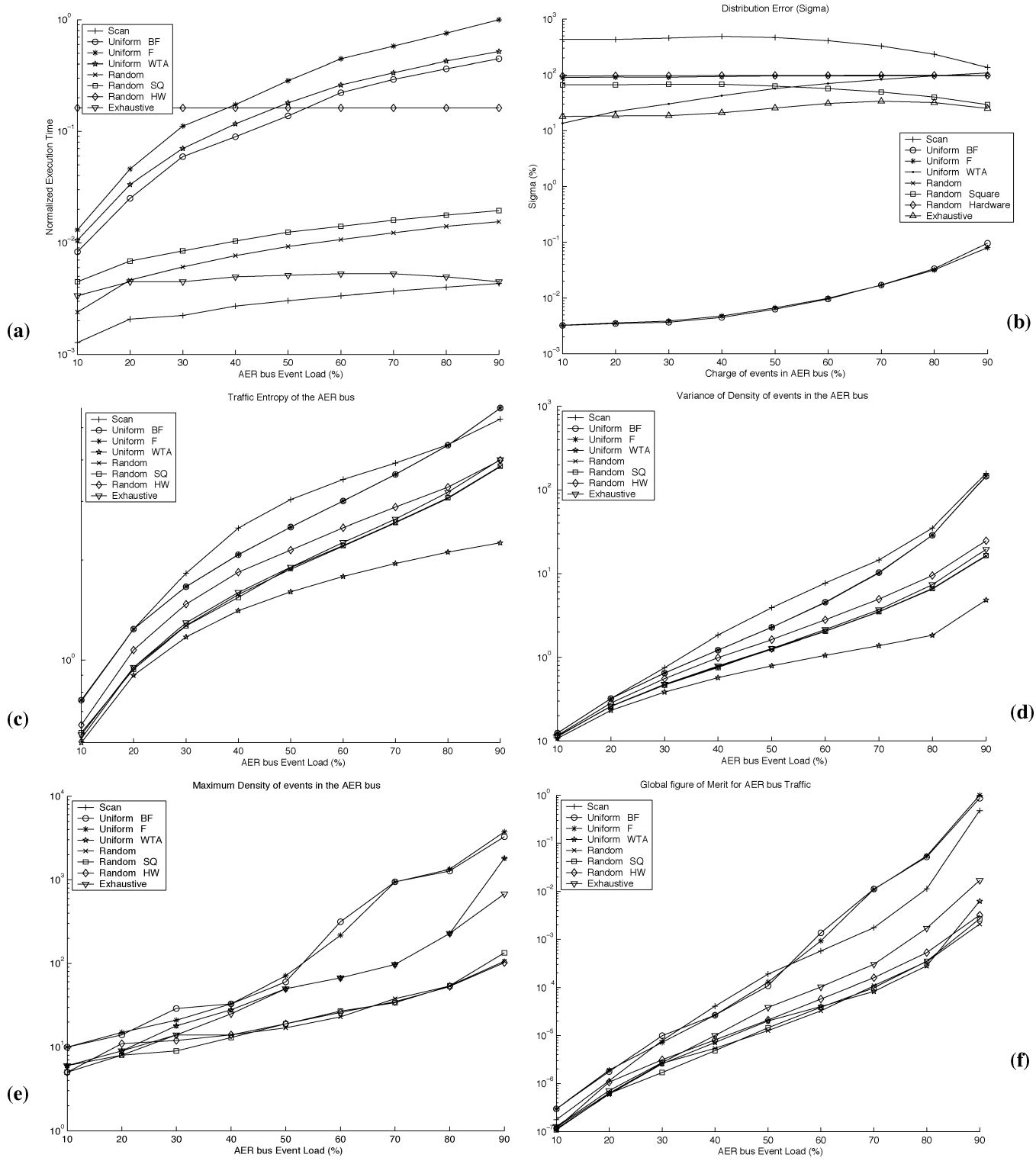


Fig. 7. Comparisons of the different algorithms. (a) Execution time comparison of software implementation for the different Synthetic AER generation Algorithms. (b) Distribution Error σ_e (in %) as defined in (7), as function of AER Traffic Load, for all synthetic AER generation methods. (c) Entropy of S set for the AER bus. (d) Variance of D for the AER bus. (e) Maximum of D for the AER frame vectors. (f) Overall clustering behavior.

speed is almost independent of event load. Algorithms *Random* and *Random-SQ* are the next fastest ones, and their speed increases between a factor of two to three, from 10% to 90% of the event load. The three *Uniform* algorithms are the ones that most depend on image load, yielding almost a factor 100 of speed difference between the cases of 10% and 90% of the event load. Algorithm *Random Hardware* has computation times completely independent of image event load, therefore showing a straight horizontal line in Fig. 7(a).

The vertical axis in Fig. 7(a) is normalized with respect to the slowest situation: 90% bus load for the *Uniform-WTA* method. This corresponds to 62.62 s of CPU time on a 1.6-GHz Pentium

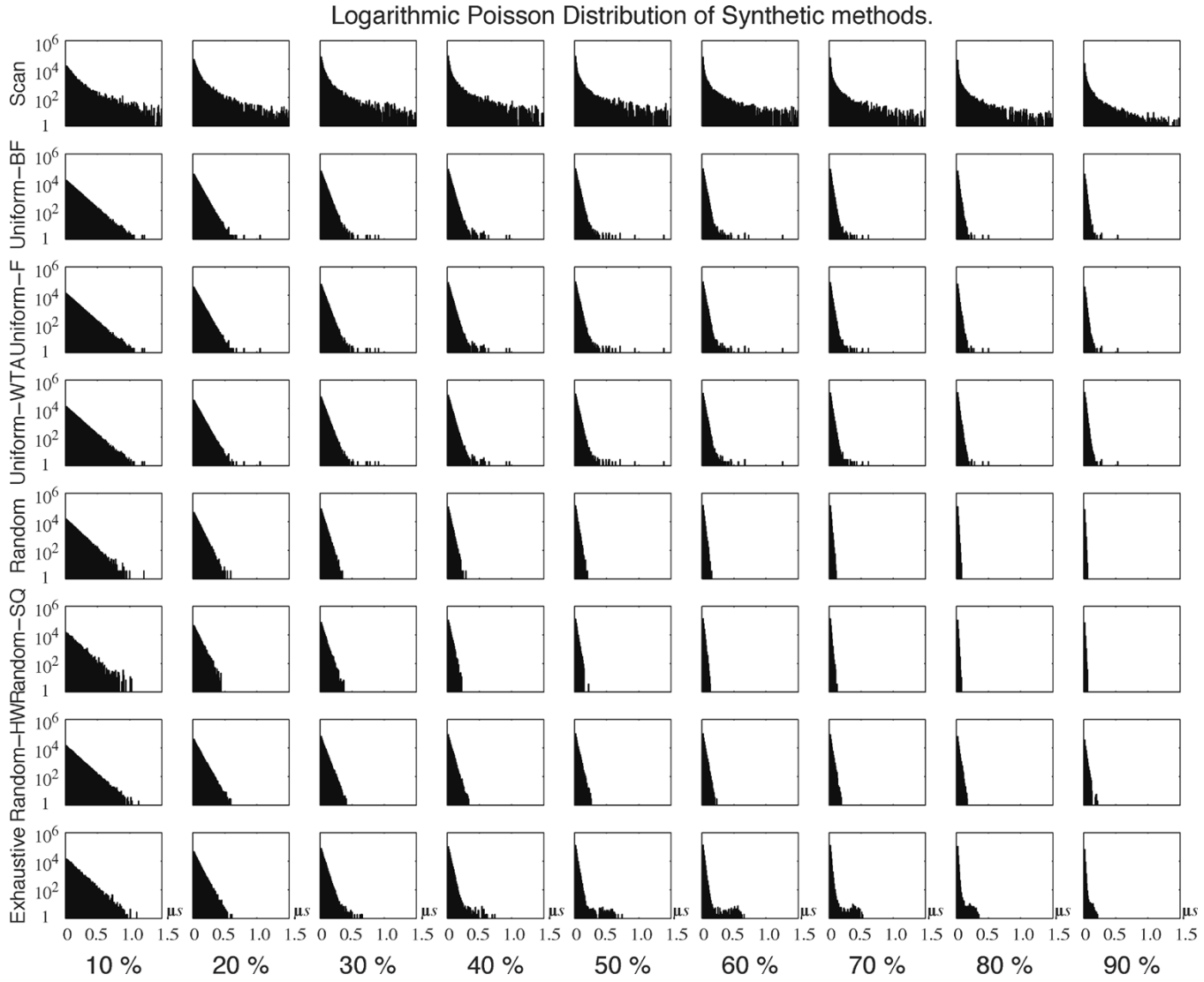


Fig. 8. Histograms of interspike times for the nine images of our TIS, using the eight synthetic AER generation algorithms. *X* axes show interspike times, and *Y* axes the number of interspike intervals within a given interspike time window. Each row of histograms corresponds to one algorithm, while each column corresponds to a TIS image.

IV computer. Consequently, since TIS images are 128 × 128 pixels, this corresponds to 3.8 ms/pixel. For a 50% bus load, the spread in computation time is from 0.011 ms/pixel to 1.14 ms/pixel, depending on the algorithm.

B. Distribution Error

In an ideal AER distribution all events for one pixel (of constant value during $T_{\rm frame}$) are equidistant in time. However, the algorithms outlined earlier will not implement exact time spacings for the events of the same pixel, because either the nature of the algorithm does not guarantee it, or because different pixels collide on the same *frame vector* slot and the algorithm looks for displaced slots or erases events. In this section the distribution of events obtained with each algorithm is evaluated. Let us call *Distribution Error* how much the event distribution generated by an algorithm deviates from the ideal distribution. First we need to provide a mathematical definition for this *Distribution Error*.

Let us call D the ideal distance between events of the same pixel of an $N \times M$ image with $K - n_{\rm res}$ gray level values. Then

$$D = \frac{n_{\rm slots}}{n_{\rm evf}} = \frac{N \times M \times K}{n_{\rm evf}}. \tag{6}$$

For events generated by algorithms the distance between events will change, in principle, from event to event for the same pixel. Let us call d_k the distance between the kth event and the $(k+1)$th one for a given pixel. For each interevent interval the error is $e_k = |D - d_k|$. Let us call *Distribution Error* the relative standard deviation of all $\{e_k\}$ for this pixel

$$\sigma_e = \frac{1}{D}\sqrt{\frac{\sum_{k=1}^{n_{\rm evf}} |D - d_k|^2}{n_{\rm evf} - 1}}. \tag{7}$$

Fig. 7(b) shows this distribution error standard deviation (in percent) for the different methods and for the nine images of our TIS. The *x* axis represents the image *event load* and the

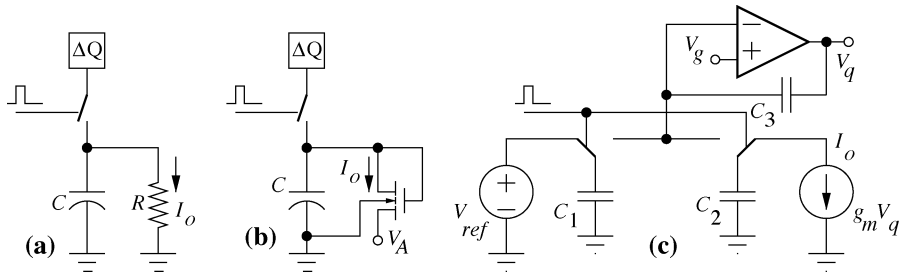


Fig. 9. Integrator circuits for AER reconstruction. (a) *RC* lowpass filter. (b) Boahen's diode-capacitor integrator. (c) Mortara's PFM demodulator.

y axis is the distribution error σ_e as defined in (7). As can be seen, the *Uniform-B* and *Uniform-BF* algorithms yield a practically negligible distribution error. For a 90% event load, the error is as low as 0.1%. The worst results, as expected, are obtained with the *Scan* algorithm, in which the distribution error standard deviation changes between 90% and 400% for our TIS. The *Uniform-WTA* algorithm behaves also much worse than the other two uniform algorithms, yielding an error standard deviation starting at 3% and increasing until 100%. The *Exhaustive* algorithm behaves relatively well, with an error that starts at 4% and has a maximum of 30%. The *Random-Square* algorithm yields a minimum error of 15% at minimum load and a maximum of 50% at a load of about 50%. Algorithms *Random* (with a 2-bit counter) and *Random-Hardware* behave almost identically. They yield a minimum of about 20% at minimum load and a maximum of about 100% at maximum load.

C. Histograms of Distributions

An interesting measure is the histogram of interspike intervals in the *frame vector*. In this case we are looking at the times between events without looking at the event addresses. That is, all addresses are considered. Such histograms have been studied before [14], [23] and Poisson-type distributions are expected for biological as well VLSI AER systems. Consequently, histograms representing in the x axis the interspike times and using a logarithmic scale for the y axis should ideally show a straight line of negative slope. We have computed the histograms for *frame vectors* generated using the different methods for the images of our TIS. Fig. 8 shows the resulting histograms. As may be seen, the *Scan Algorithm* does not provide a Poisson-like distribution. However, for the rest of the algorithms the histograms show the Poisson exponential behavior. For the *Exhaustive Algorithm* such exponential behavior is however degraded for longer interevent intervals. The *Uniform Algorithms* also show a slight degradation for longer intervals. The *Random Algorithms* do not show this degradation, although the *Random-SQ Algorithm* tends to show a truncation after a certain interevent time. In general, we may conclude that, except for the *Scan Algorithms*, all seem quite Poisson-like, especially the *Random* and the *Random-HW* algorithms.

D. Event Clustering

Here, we would like to evaluate the ability of an algorithm to avoid the formation of large clusters of events in the *frame vector*. An algorithm with this property would be more useful, for example, if we need to interface our synthetic AER generator to an AER chip (or system) of lower AER bandwidth (larger T_{event}). If our algorithm tends to cluster events (fill large number of consecutive slots), our AER sender will be forced to wait for long periods of time due to the slower AER receiver. This will distort the original distribution generated by the algorithm. On the other hand, if the algorithm has the tendency of inserting empty slots in event-dense regions, then it helps slower AER receivers by giving them extra time to catch up, and the original event distribution will be less distorted.

A useful concept that comes in handy for this purpose could be that of *Entropy*. The entropy H of a set S is defined as [47]

$$H(S) = -\sum p_i \log(p_i) \quad \text{with} \quad \sum p_i = 1 \tag{8}$$

where p_i is the probability of the elements of S. Let us generate, for our purpose, the following set $S = \{a_1, a_2, \ldots a_n\}$. Each element a_i is equal to the number of clusters of size i. Thus, if a *frame vector* has 300 groupings of 5 consecutive events (with empty slots before and after each event), then $a_5 = 300$. Note that the total number of events in the *frame vector* can be expressed as

$$n_{evT} = \sum_{i=1}^{n} i a_i \quad \Leftrightarrow \quad \sum_{i=1}^{n} \frac{i a_i}{n_{evT}} = 1. \tag{9}$$

Given a *frame vector* already generated by a given algorithm, if we pick randomly one of the n_{evT} events, the probability of it being in a cluster of size i (element a_i of S) is

$$p_i = \frac{i a_i}{n_{evT}}. \tag{10}$$

Consequently, using this p_i we can use (8) to obtain an estimate of the entropy H of set S. Low entropy values will tell us the algorithms have a tendency to form a low variety of clusters, while large entropy values reflect the fact that the algorithms tend to produce a large variety of clusters. For our purposes, we would like the algorithms to have a tendency of not forming clusters (a_i very large) or forming clusters of small size (a_i tends to zero as i increases). This implies that we want a small variety of clusters and of small size. Consequently, we are looking for algorithms that provide low values of the previously defined entropy. However, low entropy values are also produced when only large clusters are produced. In this case, we need to penalize such situations.

Let us use the following method to quantify the statistics of forming large size clusters. First a vector D_c is generated by scanning the *frame vector* once, slot by slot. Every time there

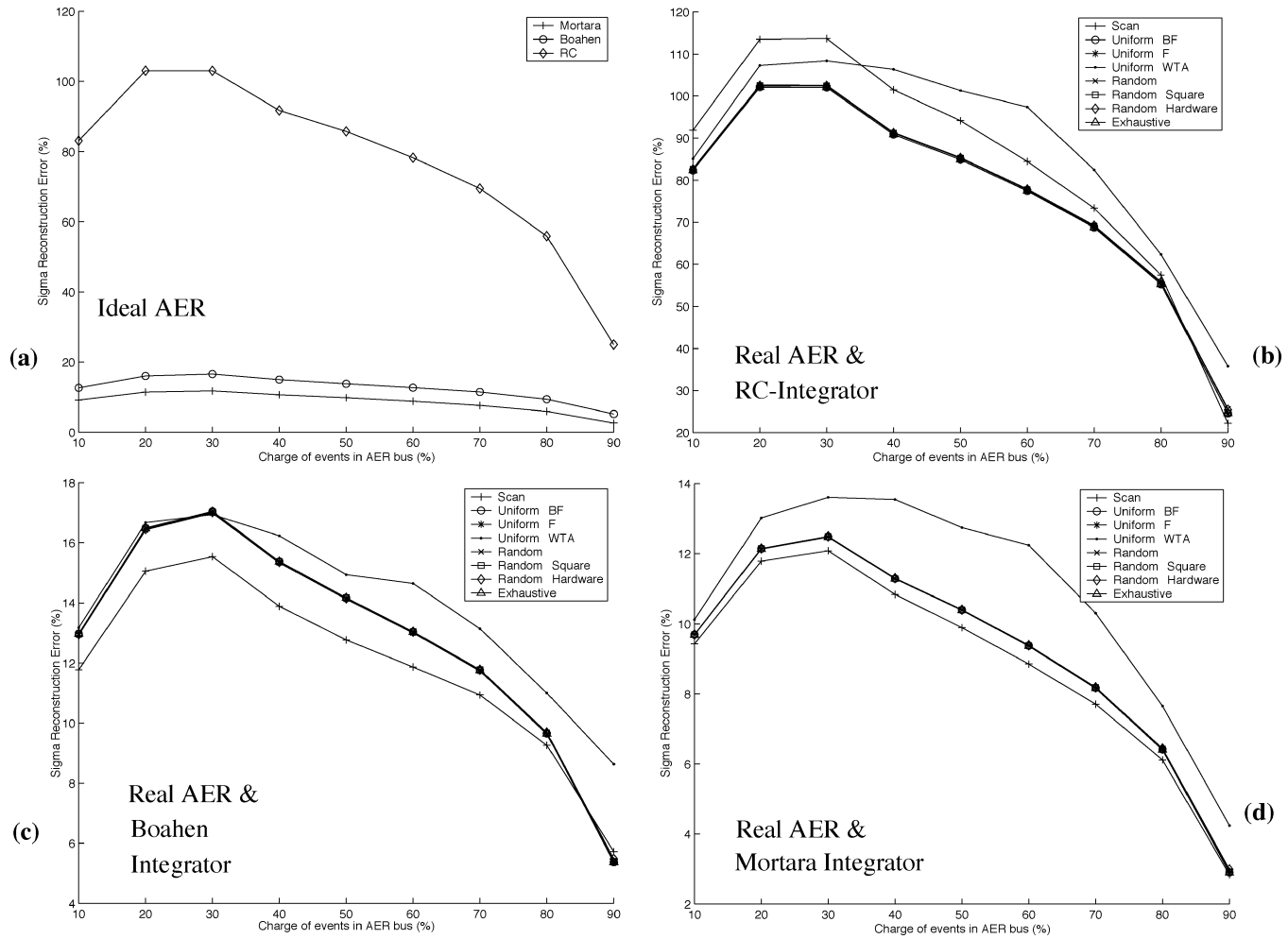


Fig. 10. Algorithm comparisons based on integrator reconstructions. (a) Reconstruction errors for the TIS images using ideal AER distribution frame vectors and the three reconstruction integrators: Mortara, Boahen, and RC. (b) Reconstruction errors for the TIS images using algorithmically generated AER distributions frame vectors and the RC reconstruction integrator. (c) Reconstruction errors for the TIS images using synthetically generated AER distributions frame vectors and the Boahen reconstruction integrator. (d) Reconstruction errors for the TIS images using synthetically generated AER distributions frame vectors and the Mortara reconstruction integrator.

is an empty slot we set $D_c(j) = 0$ and increment j by one. Every time there is a cluster a_i of size i, we set $D_c(j) = i$ and increment j by one. Obviously, $\sum D_c(j) = n_{evT}$. For vector D_c let us now compute its maximum

$$M_{D_c} = \max_j \{D_c(j)\} \tag{11}$$

and standard deviation

$$\sigma_{D_c} = \sqrt{\frac{\sum_j \left(D_c(j) - \overline{D_c}\right)^2}{\max(j) - 1}}. \tag{12}$$

Our synthetic AER generation algorithms have a tendency to form clusters of small size if our previously defined entropy H yields small values and at the same time M_{D_c} and σ_{D_c} are of low values as well. Consequently, a relevant figure of merit would be the product of the three. Fig. 7(c) shows the entropy obtained for the proposed algorithms using our TIS. Fig. 7(d) shows the resulting values of σ_{D_c}, Fig. 7(e) those for M_{D_c}, and Fig. 7(f) shows the product $H(S) \times \sigma_{D_c} \times M_{D_c}$ normalized to the maximum.

As may be seen, the three random-based methods tend to avoid clustering, while the *Uniform-BF* and *Uniform-F* have a significantly stronger tendency to form event clusters. Algorithms *Scan*, *Exhaustive*, and *Uniform-WTA* have an intermediate event-clustering behavior, surprisingly similar.

E. Reconstruction Error

In this subsection we will compare the performance of the algorithms by evaluating the images that result from reconstructing the AER stream in the frame vectors. To do so we need to use some kind of event integration mechanism. We will use the following three methods, reported and used previously in AER literature:

1) Linear Low-Pass Filter: This corresponds to the case in which an event in a receiver pixel generates a fixed charge packet ΔQ which is integrated on a capacitor C with a resistor R in parallel [48] [see Fig. 9(a)]. The current established in the resistor

TABLE I

Algorithm	Requires Frame Vector	Computational Load (software)	Event Distribution Error	Poisson Like	Event Clustering Avoidance	Reconstruction Error after Integration
Scan	NO	LOW	POOR	POOR	POOR	GOOD
Uniform-BF	YES	HIGH	Very Good	FAIR	POOR	GOOD
Uniform-F	YES	HIGH	Very Good	FAIR	POOR	GOOD
Uniform-WTA	YES	HIGH	FAIR	FAIR	FAIR	GOOD
Random	YES	FAIR	FAIR	GOOD	FAIR	GOOD
Random-SQ	YES	FAIR	FAIR	FAIR	FAIR	GOOD
Random-HW	NO	HIGH	FAIR	GOOD	FAIR	GOOD
Exhaustive	NO	LOW	GOOD	FAIR	FAIR	GOOD

is the reconstructed pixel value. After an infinite settling time, this value would be given by

$$I_o(\infty) = \frac{\Delta Q}{RC}\left(\frac{1}{e^{\frac{1}{(f_p RC)}} - 1}\right) \tag{13}$$

where the time constant RC is chosen to be such that the integrator can settle approximately within a frame period T_{frame}. If $f_p RC \gg 1$ (which is true when $n_{\text{evf}} \gg 1$), then

$$I_o(\infty) \approx \Delta Q f_p. \tag{14}$$

2) Boahen Diode-Capacitor Integrator: The previous integrator requires a resistor R and capacitor C to achieve a time constant in the order of T_{frame}. In VLSI microchip technology the size of the resistor and capacitor would be extremely large for such low time constants [49]. Since an RC pair is required for each pixel, this approach does not result practical. Boahen's diode-capacitor integrator, shown in Fig. 9(b), provides an integration value which is also linear with event frequency [19]

$$I_o(\infty) = A(\Delta Q) f_p \tag{15}$$

(where A is a constant gain factor controlled by the MOS source voltage V_A), requiring also much less area in standard VLSI technology [19].

3) Mortara PFM Demodulator: Another AER reconstruction mechanism reported by Mortara *et al.* is the pulse-frequency-modulation (PFM) demodulator, shown in Fig. 9(c) [10], [50]. This demodulator is more complex than the two previous integrators. However, it has the nice feature of keeping a constant interevent output, while the other two integrators discharge slowly in the absence of events producing a rippled output signal (that is, with extra noise). The steady-state output signal for this circuit is given by

$$I_o(\infty) = (V_{\text{ref}} - V_g) C_1 f_p. \tag{16}$$

We have behaviorally modeled the three previous reconstruction integrator circuits in MATLAB, using ideal circuit component descriptions that yield sets of finite difference equations [44]. First we used an ideal *frame vector* to compute the integrator circuits outputs at time T_{frame}. By ideal *frame vector* we mean the resulting ideal event distribution, in which each event can be allocated where it should be theoretically and no collisions occur (this limit situation corresponds to an ideal zero event width $T_{\text{event}} = 0$ and infinite number of slots ($n_{\text{slots}} = \infty$) for a single frame. Under these circumstances the events of a single pixel are equally spaced during the frame time. Testing the resulting ideal *frame vectors* for our TIS and the previous three integrator circuits we computed the errors between the original images in the TIS and the reconstructed images at time T_{frame}.

It can be shown that for Boahen's integrator the longest settling time[7] occurs when n_{evf} changes from maximum (K) to minimum (1). In this case, the settling time is approximately given by $(K^2(\ln 3))/f_{\max}$ [44]. If we are assigning one event per frame for the lowest activity then $f_{\max} = K/T_{\text{frame}}$, which gives a worst-case settling time of $K(\ln 3) \times T_{\text{frame}}$. In this respect, Mortara's integrator is more efficient because its worst-case settling time (which happens for a transition from minimum (1) to maximum (K)) is approximately $(2K(\ln K))/f_{\max}$ or $2\ln(K) \times T_{\text{frame}}$ [44]. In our case, we would like to estimate the reconstruction error after a time T_{frame}, and how the different synthetic AER generation algorithms degrade this error for the previous integrators with respect to an ideal AER sequence.

Fig. 10(a) shows the reconstruction errors after a time T_{frame}, using ideal AER distributions (no event reallocations because of collisions) for the different integrators, using the images of our TIS and making the integrators settle from the initial conditions set at maximum value. The *y* axis is the standard deviation, over all the pixels of an image, of the pixel errors obtained as follows (pixels with null values are excluded): Computing the difference between the steady-state final value (for $t \to \infty$) and the value at $t = T_{\text{frame}}$, normalized with respect to the steady-state value.

Now, doing exactly the same behavioral simulations, but using the *frame vectors* generated by the different algorithms, yields the results shown in Fig. 10(b) for the *RC* integrator, in

[7]Settling times are defined, for a given resolution $n_{\text{res}} = K$, as the time needed for settling to an error equivalent to the resolution.

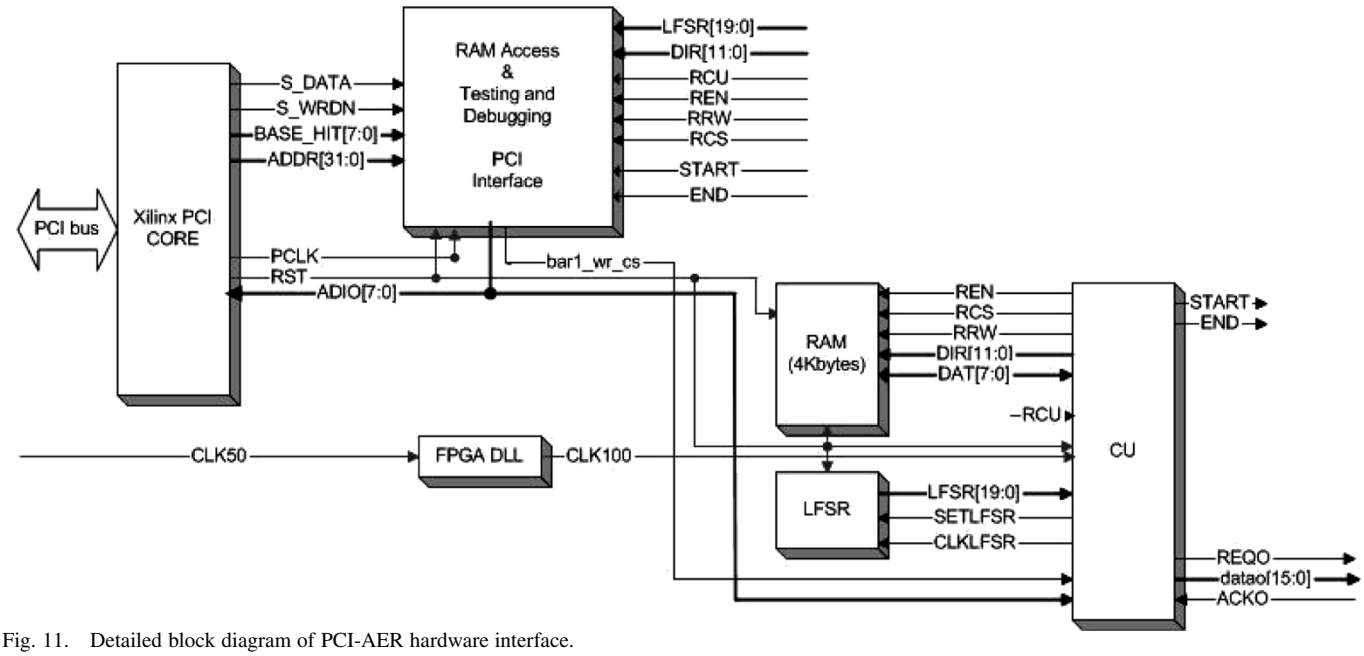


Fig. 11. Detailed block diagram of PCI-AER hardware interface.

Fig. 10(c) for Boahen's integrator, and in Fig. 10(d) for Mortara's integrator. These figures show the standard deviation of the pixel errors obtained as follows (pixels with null values are excluded): Computing the relative difference between the pixel value at $t = T_{\text{frame}}$ when using the synthetic AER generation method and when using the ideal distributions [as in Fig. 10(a)]. Comparing all three figures we may conclude that Mortara's integrator is slightly more sensitive to the distribution errors of the different methods, than the other integrators. The less sensitive one is Boahen's. With respect to the algorithmic AER generation methods, we may conclude that for low event loads there is no significant difference between methods, while for loads of 40% or higher it is better not to use the *Uniform-WTA* nor the *Scan* algorithms. This is because the former discards colliding events and the latter groups them in a nonnatural way. Regarding reconstruction with integrating circuits, the conclusion is that there is no significant difference between the methods used for AER generation. On the other hand, we have observed differences depending on the integrators used.

Table I summarizes the comparison results of this section.

IV. Hardware Interface

All analyzes and results discussed so far have been performed in software. However, our ultimate goal is to be able to provide hardware solutions for the synthetic AER generation in real time. We have built a computer-to-AER interface that uses the standard PCI computer bus. The interface includes the communication circuitry to the computer PCI bus, the transformation of video streams from FBR to AER using the *Random-HW Algorithm*, and the asynchronous AER communication to conventional AER buses. This hardware interface has been coded in VHDL, synthesized into a VirtexE 300 FPGA [52], and tested on a Nallatech Ballyinx prototyping board [53]. The interface can write AER events every $T_{\text{event}} = 100$ ns. Consequently, this implies the restriction that $N \times M \times K \leq T_{\text{frame}}/T_{\text{event}} = 40\ \text{ms}/100\ \text{ns} = 4 \times 10^5$. As a result, this hardware is able to generate AER in real-time for a 64×64 pixels image ($N = M = 64$) at a maximum rate of $K = 255$ events per pixel. The hardware interface is capable of generating a peak rate of 10^7events/s.

Fig. 11 shows the architecture of the hardware interface. It includes a PCI core, a 4 KB RAM for storing a 64×64 frame, a *control unit* (CU), a 20-bit *linear-feedback-shift-register* (LFSR), a delay-line-loop (DLL) for internal clock management, a decoder and a set of control registers for managing the PCI core and configuring the rest of components. The image of a 64×64 pixels frame is transferred from the computer through its PCI bus and the PCI core to the 4 KB RAM memory. The 20-bit LFSR is used for the pseudorandom number generation and is the core of the *Random-HW* AER generation algorithm. The 12 less significant bits are pixel addresses, while the 8 most significant ones are compared against the pixel value. The CU, clocked at 100 MHz, is the operation center. It manages the RAM access, and this is done in two ways: a) Through the PCI bus for loading frames to be converted into AER, and b) through the 12 less significant bits of the LFSR to address a frame pixel and decide whether or not to send an event with its address. The LFSR works at a slower speed, using a clock which is generated by the CU and triggered by the communication with the AER receiver.

As an illustration of the hardware interface operation, a 64×64 pixel version of the image in Fig. 2(a) was fed to the hardware interface and the generated events were captured. The image was generated setting $T_{\text{frame}} = 40$ ms and $K = 255$. Events were captured using a logic analyzer, and then reconstructed off-line by a computer. The top row in Fig. 12 shows the result of processing directly the image in Fig. 2(a), while the bottom row corresponds to the same image after edge extraction using a Sobel Filter [54]. Each image in Fig. 12 has been reconstructed by using only the first fraction of events of the complete frame. For example, in the top row left image we used only the first 1% of the events of the whole frame to reconstruct the image. Also

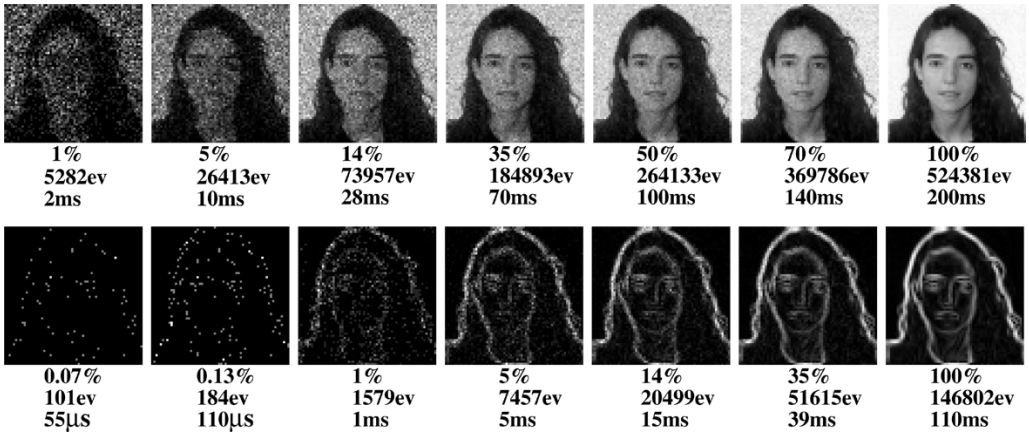


Fig. 12. Reconstructed images from events obtained from the Hardware AER output of Fig. 11. Top row: Intensity coded image. Bottom row: Same image after Sobel Edge Extraction Filter. The numbers below each frame indicate: (1) Fraction of total frame events used for reconstruction, (2) absolute number of events used, and (3) absolute time used for collecting this fraction of events.

shown are the resulting images of using only the first 5%, 14%, 35%, 50%, 70% events, and also all 100% events of the frame. In this case, the complete 100% frame was represented by approximately 524 Kevents (26% of AER bus limit: Two Mevents at 100 ns/event during 200 ms). As can be seen, it is possible to recognize the image before all events have been transmitted. This is a very interesting property of AER:The most relevant pixels (the one with higher intensity) are transmitted first, and consequently present a given space-time correlation. For the bottom row in Fig. 12, this observation is even more impressive. In this case, we are transmitting the computed edges of the previous image. The resulting AER coding will be much more sparse than before. In this case, the complete 100% frame was represented by 147 Kevents (13% of AER bus limit: 1.1 Mevents at 100 ns/event during 110 ms). As can be seen, with only the first 0.07%–0.13% of events one can already recognize the silhouette.

These figures illustrate how the Random-HW algorithm adds artificial noise to the images, as a consequence of its random nature (unless the original and reconstructed frames are perfectly synchronized, as in the top right of Fig. 12; however, this situation is usually not met in practice).

In order to analyze how the hardware generated streams differ from the ones generated directly by software, we compared both according to the same criteria discussed in Section III. The results are shown in Fig. 13. Fig. 13(a) shows the differences in distribution error, as defined by (7), for the events of the top-right frame in Fig. 12 obtained from our hardware and reproducing the same conditions in software. The negligible differences between the hardware and software event distributions are caused by extra delays of the hardware not modeled in the software. Note that, compared to what is shown in Fig. 7(b) for the *Random-HW* method (with 128×128 images), there is no significant difference. Fig. 13(b) summarizes the event clustering analysis [as in Fig. 7(c)–(f)]. Continuous lines are for software simulations, while dotted lines correspond to our hardware results. Crosses indicate the *entropy* as defined in (8)–(10). Circles indicate the maximum values of the D_c vector [see (11)]. Stars show the values for the standard deviations of the D_c vector [see (12)]. There is almost no appreciable difference between the SW and HW events streams. Dots show the product of all the previous three characteristics.

In Fig. 13(c) we show the interspike time histograms. As may be seen, the difference between the hardware (captured) and software (simulated) histograms are quite minor.

V. Conclusion

Eight different methods for transforming a FBR video stream to a rate-coded AER one are presented. The methods can be grouped into four types: Scan, uniform, random, and exhaustive. The methods have been implemented and tested in software, and compared according the different criteria, and for different AER traffic loads. Uniform methods result in the most precise distributions but at the expense of costly computations. The scan method is the fastest one but also the most imprecise. The exhaustive method is a good compromise between speed and precision. Random methods also show a good speed/precision tradeoff, although they add more noise. Random methods yield good Poisson distributed events. Scan and uniform methods have a tendency to form clusters. All methods except three (the scan method, one of the random methods, and the exhaustive method), require the use of a large memory frame vector buffer. The random method that does not require this buffer has been implemented in hardware using a prototyping board with a VirtexE 300 FPGA. The AER generated by this hardware was captured using a logic analyzer. Thus, the hardware has been tested against its corresponding software behavior, and no significant differences were observed. Experimentally obtained AER sequences were reconstructed off-line and are shown in the paper. Future work will concentrate on nonrate-coded AER generation.

Appendix

The FBR-to-AER algorithms perform a transformation from pixel activity p to pixel event frequency f_p. Let us assume pixel activity is constrained to a normalized unity interval $p \in [0, 1]$. We want the algorithms to map this activity interval into an event frequency interval $f_p \in [0, f_{\max}]$, and we want a linear transformation ($f_p = f_{\max} P$) as a VLSI AER system [10]–[27] would

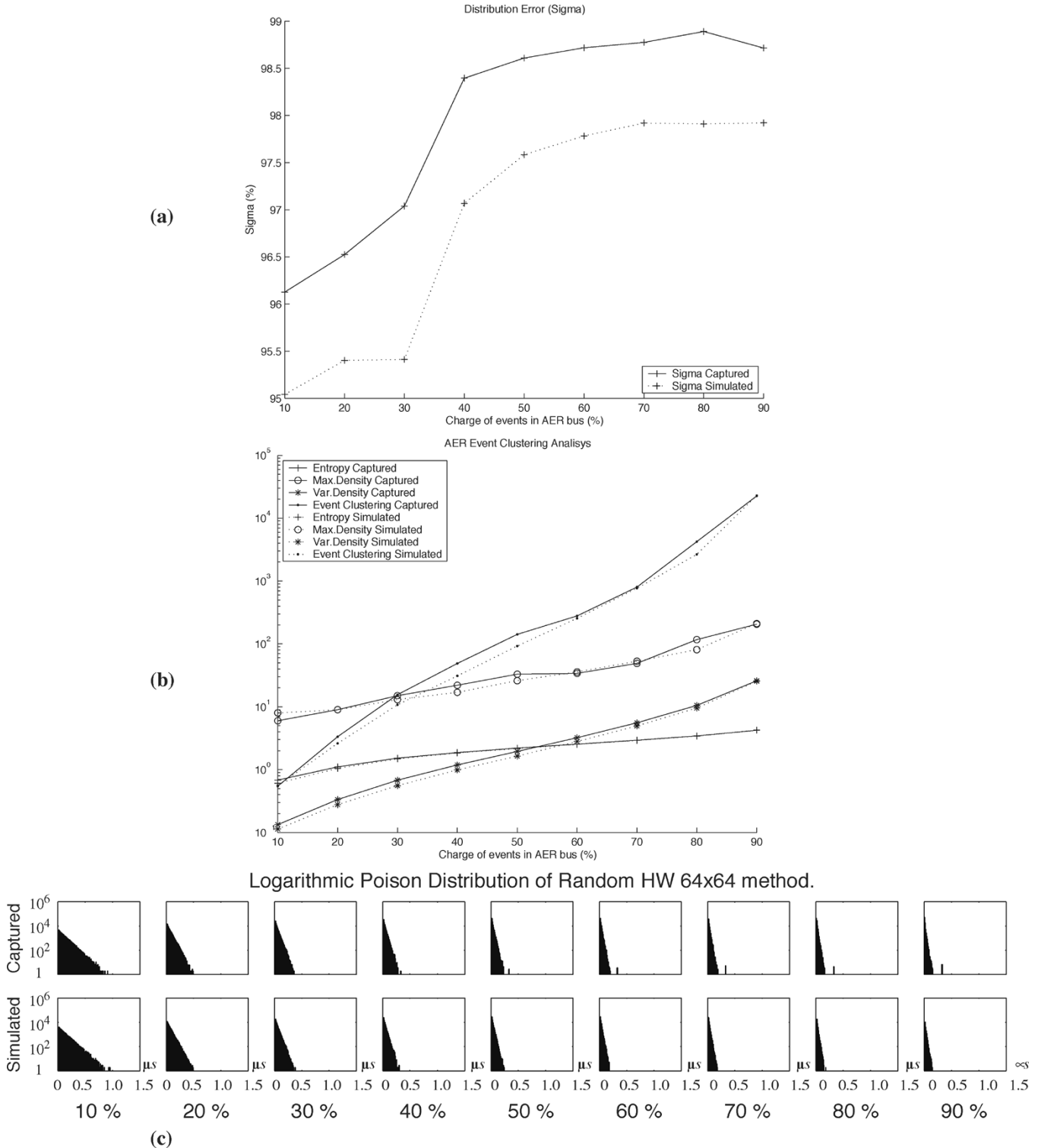


Fig. 13. Comparison of HW and SW generated streams. (a) Distribution error as defined in (7). (b) Event clustering analysis. (c) Poisson distribution analysis.

do it. For zero pixel activity the output frequency should be zero, and no events are transmitted. However, in general we would like to transmit events only if a minimum activity is reached. Let us call $f_{\min}$ the frequency that corresponds to the minimum nonzero pixel activity we want to transmit. Consequently, we will impose a nonzero $f_{\min}$. Under this constraint, the extreme frequencies $f_{\min}$ and $f_{\max}$ will be related by a finite positive proportionality constant K ($f_{\max} = Kf_{\min}$). Since we want to do the FBR-AER transformations frame by frame, we need to make $f_{\min} \geq 1/T_{\text{frame}}$, or equivalently $f_{\min} = a/T_{\text{frame}} (a \geq 1)$, where T_{frame} is the time assigned to one frame. For the limit situation $a = 1$, one obtains $f_p = (Kp)/T_{\text{frame}}$. Another constraint is the following. Frame images proceed from a conventional digital video system or computer. Consequently, pixel intensities will take discrete values in the $p \in [0, 1]$ interval. In case of 8-bit luminance coding, for example, the interval would have a resolution of $n_{\text{res}} = 2^8 = 256$ levels. The minimum increment in p is therefore $\Delta p = 1/n_{\text{res}}$. Let

us now express the number of events to be transmitted during the time interval of a frame $T_{\rm frame}$ for a pixel of activity p. This is the "number of events per frame" for this pixel $n_{\rm evf} = f_p T_{\rm frame} = Kp$. Since p is of discrete nature with steps of size $\Delta p = 1/n_{\rm res}$, then $n_{\rm evf}$ will also be of discrete nature with steps of size $\Delta n_{\rm evf} = K\Delta p = K/n_{\rm res}$. From a practical point of view, it is desirable that the number of events per frame $n_{\rm evf}$ to be generated is integer. Otherwise, one would have to generate different number of events for consecutive frames for the same pixel activity.[8] This would require us to use some kind of memory from frame to frame, which is a complication we would like to avoid. We want to do the FBR-AER transformation frame by frame. Consequently, $n_{\rm evf}$ will be integer and so will be $\Delta n_{\rm evf}$. Since $n_{\rm res}$ is integer, K will be integer as well, and also a multiple of $n_{\rm res}$. Furthermore, if we decide to make $\Delta n_{\rm evf} = 1$, then $K = n_{\rm res}$. We have constrained ourselves to this situation throughout the paper, independently of the method used for the FBR-AER transformation. However, we should keep in mind that this constraint is arbitrary (it is a particular case of $a = \Delta n_{\rm evf} = 1$). Consequently, in this paper, we can consider that the pixels of the frames store the value $n_{\rm evf} = Kp = n_{\rm res}p$ directly. Therefore

$$f_p = p\frac{n_{\rm res}}{T_{\rm frame}}. \tag{17}$$

ACKNOWLEDGMENT

The authors would like to thank R. Senhadji-Navarro, F. Gómez-Rodríguez, F. Olmedo, and I. García-Vargas for their helpful discussions. Special thanks to I. García-Vargas for proposing the *Exhaustive Algorithm*.

REFERENCES

[1] G. M. Shepherd, *The Synaptic Organization of the Brain*, 3rd ed. Oxford, U.K.: Oxford Univ. Press, 1990.
[2] T. Crimmins, "Geometric filter for speckle reduction," *Appl. Opt.*, vol. 24, pp. 1438–1443, 1985.
[3] S. Grossberg, E. Mingolla, and J. Williamson, "Synthetic aperture radar processing by a multiple scale neural system for boundary and surface representation," *Neural Netw.*, vol. 8, no. 7/8, pp. 1005–1028, 1995.
[4] S. Grossberg, E. Mingolla, and W. D. Ross, "Visual brain and visual perception: how does the cortex do perceptual grouping?," *Trends in Neurosci.*, vol. 20, pp. 106–111, 1997.
[5] J. Lee, "A simple speckle smoothing algorithm for synthetic aperture radar images," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-13, pp. 85–89, 1983.
[6] E. Mingolla, W. Ross, and S. Grossberg, "A neural network for enhancing boundaries and surfaces in synthetic aperture radar images," *Neural Netw.*, vol. 12, no. 3, pp. 499–511, 1999.
[7] M. Sivilotti, "Wiring considerations in analog VLSI systems with application to field-programmable networks," Ph.D. dissertation, Calif. Inst. Technol., Pasadena, 1991.
[8] M. Mahowald, "VLSI analogs of neural visual processing: a synthesis of form and function," Ph.D. dissertation, Calif. Inst. Technol., Pasadena, 1992.
[9] M. Mahowald, *An Analog VLSI Stereoscopic Vision System*. Boston, MA: Kluwer Academic, 1994.
[10] A. Mortara and E. A. Vittoz, "A communication architecture tailored for analog VLSI artificial neural networks: intrinsic performance and limitations," *IEEE Trans. Neural Netw.*, vol. 5, no. 3, pp. 459–466, 1994.
[11] A. Mortara, E. A. Vittoz, and P. Venier, "A communication scheme for analog VLSI perceptive systems," *IEEE J. Solid-State Circuits*, vol. 30, no. 6, pp. 660–669, Jun. 1995.

[8]For example, if one needs to generate $n_{\rm evf} = 1.5$, then in one frame one can produce one event, in the next two events, one more in the next one, and so on.

[12] P. Vernier, A. Mortara, X. Arreguit, and E. A. Vittoz, "An integrated cortical layer for orientation enhancement," *IEEE J. Solid-State Cicuits*, vol. 32, pp. 177–186, Feb. 1997.
[13] J. Kramer, R. Sarpeshkar, and C. Koch, "Pulse-based analog velocity sensors," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 44, pp. 86–101, 1997.
[14] E. Culurciello, R. Etienne-Cummings, and K. Boahen, "A biomorphic digital image sensor," *IEEE J. Solid-State Circuits*, vol. 38, no. 2, pp. 281–294, Feb. 2003.
[15] T. Serrano-Gotarredona, A. G. Andreou, and B. Linares-Barranco, "AER image filtering architecture for vision-processing systems," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 46, no. 9, pp. 1064–1071, Sep. 1999.
[16] R. Serrano-Gotarredona, T. Serrano-Gotarredona, A. Acosta-Jiménez, B. Linares-Barranco, C. Serrano-Gotarredona, A. Linares-Barranco, G. Jiménez-Moreno, and A. Civit-Ballcels, "An arbitrary kernel convolution AER transceiver," *Proc. IEEE Int. Symp. Circ. Syst. (ISCA'06)*, May 2006.
[17] J. Lazzaro, J. Wawrzynek, M. Mahowald, M. Sivilotti, and D. Gillespie, "Silicon auditory processors as computer peripherals," *IEEE Trans. Neural Netw.*, vol. 4, no. 3, pp. 523–528, May 1993.
[18] A. Abusland, T. S. Lande, and M. Hovin, "A VLSI communication architecture for stochastically pulse-encoded analog signals," in *Proc. 1996 IEEE Int. Symp. Circuits Syst., ISCAS'96*, vol. III, Atlanta, GA, May 1996, pp. 401–404.
[19] K. Boahen, "Retinomorphic vision systems," in *Proc. 5th Int. Conf. Microelectron. Neural Netw. Fuzzy Syst.*, Lausanne, Switzerland, Feb. 1996, pp. 2–14.
[20] K. Boahen, "The retinomorphic approach: pixel-parallel adaptive amplification, filtering and quantization," *Int. J. Analog Integr. Circuits Signal Process.*, vol. 13, pp. 53–68, May/Jun. 1997.
[21] K. Boahen, "Communicating neuronal ensembles between neuromorphic chips," in *Neuromorphic Systems Engineering: Neural Networks in Silicon*. Boston, MA: Kluwer Academic, 1998, ch. 11, pp. 229–262.
[22] K. Boahen, "A throughput-on-demand address-event transmitter for neuromorphic chips," in *1999 Conf. Adv. Res. VLSI*, Atlanta, GA, 1999, pp. 72–86.
[23] K. Boahen, "Point-to-point connectivity between neuromorphic chips using address events," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 47, no. 5, pp. 416–434, May 2000.
[24] S. R. Deiss, R. Douglas, and A. A. Whatley, "A pulse-coded communications infrastructure for neuromorphic systems," in *Pulsed Neural Networks*, W. Maass and C. M. Bishop, Eds. Boston, MA: MIT Press, 1999, ch. 6, pp. 157–178.
[25] Z. Kalayjian, J. Waskiewicz, D. Yochelson, and A. G. Andreou, "Asynchronous sampling of 2D arrays using winner-takes-all arbitration," in *Proc. 1996 IEEE Int. Symp. Circuits Syst. (ISCAS'96)*, vol. 3, Atlanta, GA, 1996, pp. 393–396.
[26] J. P. Lazzaro and J. Wawrzynek, "A multi-sender asynchronous extension to the address-event protocol," in *16th Conf. Adv. Res. VLSI*, W. J. Dally, J. W. Poulton, and A. T. Ishii, Eds., 1995, pp. 158–169.
[27] J. T. Marienborg, T. S. Lande, A. Abusland, and M. Hovin, "An analog approach to 'neuromorphic' communication," in *Proc. 1996 IEEE Int. Symp. Circuits Syst., ISCAS'96*, vol. 3, Atlanta, GA, May 1996, pp. 397–400.
[28] E. Vittoz, "Present and future industrial applications of bio-inspired VLSI systems," in *7th Int. Conf. Microelectron. Neural, Fuzzy, and Bio-Inspired Syst., MICRONEURO'99*, Granada, Spain, Apr. 1999, pp. 2–11.
[29] D. H. Goldberg, G. Cauwenberghs, and A. G. Andreou, "Analog VLSI spiking neural network with address domain probabilistic synapses," in *Proc. 2001 IEEE Int. Symp. Circuits Syst., (ISCAS 2001)*, vol. 2, May 2001, pp. 241–244.
[30] P. Häfliger, "Asynchronous event redirecting in bio-inspired communication," in *Proc. 2001 Int. Conf. Electron., Circuits Syst. (ICECS 2001)*, vol. 1, Sep. 2001, pp. 87–90.
[31] T. Y. W. Choi, B. E. Shi, and K. A. Boahen, "An on-off orientation selective address event representation image transceiver chip," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 51, no. 2, pp. 342–353, Feb. 2004.
[32] E. Culurciello and A. G. Andreou, "16 × 16 pixel silicon on sapphire CMOS digital pixel photosensor array," *Electron. Lett.*, vol. 40, no. 1, pp. 66–68, Jan., 8th 2004.
[33] A. Cohen, R. Etienne-Cummings, T. Horiuchi, G. Indiveri, S. Shamma, R. Douglas, C. Koch, and T. Sejnowski, "Rep. 2004 Workshop on Neuromorphic Eng.,", Telluride, CO, Jun. 27 to Jul. 17 2004.
[34] J. Lee and B. Razavi, "A 40-Gb/s clock and data recovery circuit in 0.18 mm CMOS technology," *IEEE J. Solid-State Circuits*, vol. 38, no. 12, pp. 2181–2190, Dec. 2003.

[35] M. Abeles, *Corticonics. Neural Circuits of the Cerebral Cortex*. Cambridge, U.K.: Cambridge Univ. Press, 1991.
[36] K. Hynna and K. Boahen, "Space-rate coding in an adaptive silicon neuron," *Neural Netw.*, vol. 14, pp. 645–656, 2001.
[37] W. Maass and T. Natschlager, "A model of fast analog computation based on unreliable synapses," *Neural Comput.*, vol. 12, no. 7, pp. 1679–1704, 2000.
[38] W. Gerstner, "Spiking neurons," in *Pulsed Neural Networks*, W. Maas and C. M. Bishop, Eds. Cambridge: MIT Press, 1999, ch. 1, pp. 3–53.
[39] S.-C. Liu and R. Douglas, "Temporal coding in a silicon network of integrate-and-fire neurons," *IEEE Trans. Neural Netw.*, vol. 15, no. 5, pp. 1305–1314, Sep. 2004.
[40] S. Thorpe, F. Fize, and C. Marlot, "Speed of processing in the human visual system," *Nature*, vol. 381, pp. 520–522, Jun. 1996.
[41] L. Perrinet, M. Samuelides, and S. Thorpe, "Coding static natural images using spiking event times: do neurons cooperate?," *IEEE Trans. Neural Netw.*, vol. 15, no. 5, pp. 1164–1175, Sep. 2004.
[42] M. Lengyel and P. Erdi, "Theta-modulated feedforward network generates rate and phase coded firing in the entorhino-hippocampal system," *IEEE Trans. Neural Netw.*, vol. 15, no. 5, pp. 1092–1099, Sep. 2004.
[43] E. M. Izhikevich, "Which model to use for cortical spiking neurons?," *IEEE Trans. Neural Netw.*, vol. 15, no. 5, pp. 1063–1070, Sep. 2004.
[44] A. Linares-Barranco, "Study and evaluation of AER interfaces for neuromorphic systems," Ph.D. dissertation, Univ. Seville, Spain, 2003.
[45] A. Linares-Barranco, R. Senhadji-Navarro, I. García-Vargas, F. Gómez-Rodríguez, G. Jimenez, and A. Civit, "Synthetic generation of address-event for real-time image processing," in *Proc. 9th Conf. Emergent Technol. Factory Automation*, vol. 2, Lisbon, Spain, Sep. 2003, pp. 462–467.
[46] S. W. Golomb, *Shift Register Sequences*. Laguna Hills, CA: Aegean Park, 1982.
[47] A. B. Carlson, P. B. Crilly, and J. C. Rutledge, *Communication Systems, an Introduction to Signals and Noise in Electrical Communications*. New York: McGraw-Hill, 2002.
[48] A. B. Apsel and A. G. Andreou, "Analysis of data reconstruction efficiency using stochastic encoding and an integrating receiver," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 48, no. 10, pp. 890–897, Oct. 2001.
[49] A. S. Sedra and K. C. Smith, *Microelectronic Circuits*. New York: Oxford Univ. Press, 1998.
[50] A. Mortara and E. A. Vittoz, "A 12-transistor PFM demodulator for analog neural networks communication," *IEEE Trans. Neural Netw.*, vol. 6, no. 5, pp. 1280–1283, Sep. 1995.
[51] K. W. Boahen and A. G. Andreou, "A contrast sensitive silicon retina with reciprocal synapses," *Adv. Neural Info. Process. Syst.*, vol. 4, pp. 764–772, 1992.
[52] *Virtex 2.5 V Field Programmable Gate Arrays Data Sheet*, Xilinx, April 2001.
[53] *Ballyinx pci64 Prototyping Board User's Guide*, Nallatech Ltd., 2001.
[54] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, Second ed. Englewood Cliffs, NJ: IE Prentice-Hall, 2002.

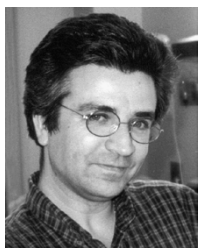
Alejandro Linares-Barranco received the B.S. degree in computer engineering, the M.S. degree in industrial computer science, and the Ph.D. degree in computer science (specializing in computer interfaces for bioinspired systems) from the University of Seville, Seville, Spain, in 1998, 2002, and 2003, respectively.

From January 1998 to June 1998, he was Second Lieutenant in the Spanish Air Force working as System Administrator and a software developer. During 1998, he also worked with the Colors Digital Communications S.L. Company, Seville. From November 1998 to February 2000, he was a Member of the Technical Staff at the Seville Microelectronics Institute (IMSE), an institute of the National Microelectronics Center (CNM), Spanish Research Council (CSIC), Seville. From March 2000 to February 2001, he was a Development Engineer with the Research and Development Department, SAINCO Company, Seville, working on VHDL-based field-programmable gate array (FPGA) systems for the INSONET European project on power line communications. Since March 2003, he has been an Assistant Professor of Computer Architecture and Technology at the University of Seville. His research interests include VLSI and FPGA digital design, vision processing systems, bus emulation, and computer architectures.

Gabriel Jimenez-Moreno received the M.S. degree in physics (electronics) and the Ph.D. degree from the University of Seville, Seville, Spain, in 1990 and 1992, respectively.

After working with Alcatel, he was granted a Fellowship from the Spanish Science and Technology Commission (CICYT). Currently, he is an Associate Professor of computer architecture at the University of Seville. From 1996 until 1998, he was Vice-Dean of the E.T.S. Ingenieria Informatica, University of Seville. He participated in the creation of the Department of Computer Architecture (also at University of Seville) and since 2004, has been its Secretary. He is the author of various papers and research reports on robotics, rehabilitation technology, and computer architecture. He has directed two national research projects on neuromorphic systems. His research interests include neural networks, vision processing systems, embedded systems, computer interfaces, and computer architectures.

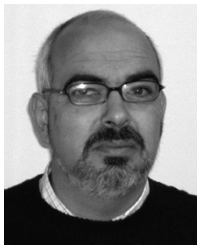
Dr. Jimenez-Moreno was Assistant Editor of the *Journal of Computer and Software Engineering* (Ablex Publishing Corp.) from 1993 to 1995. He has been Counselor of the IEEE Student Branch at the University of Seville since 2003.



Bernabé Linares-Barranco received the B.S. degree in electronic physics in June 1986 and the M.S. degree in microelectronics in September 1987, both from the University of Seville, Seville, Spain. He received a first Ph.D. degree in high-frequency OTA-C oscillator design in June 1990 from the University of Seville, and a second Ph.D. degree in analog neural network design in December 1991 from Texas A&M University, College Station.

Since September 1991, he has been a Tenured Scientist with the Sevilla Microelectronics Institute (IMSE), which is one of the institutes of the National Microelectronics Center (CNM) of the Spanish Research Council (CSIC) of Spain. In January 2003, he was promoted to Tenured Researcher, and in January 2004, to Full Professor of Research. From September 1996 to August 1997, he was on sabbatical stay with the Department of Electrical and Computer Engineering of the Johns Hopkins University, Baltimore, MD, as a Postdoctoral Fellow. During spring 2002, he was a Visiting Associate Professor with the Electrical Engineering Department, Texas A&M University. He has been involved with circuit design for telecommunication circuits, VLSI emulators of biological neurons, VLSI neural-based pattern recognition systems, hearing aids, precision circuit design for instrumentation equipment, bioinspired VLSI vision processing systems, transistor parameters mismatch characterization, address-event-representation (AER) VLSI, RF circuit design, and real-time vision processing chips. He is coauthor of *Adaptive Resonance Theory Microchips*.

Dr. Linares-Barranco was corecipient of the 1997 IEEE TRANSACTIONS ON VLSI SYSTEMS Best Paper Award for the paper "A real-time clustering microchip neural engine," and of the 2000 IEEE CAS Darlington Award for the paper "A general translinear principle for subthreshold MOS transistors." He organized the 1994 Nips Post-Conference Workshop "Neural Hardware Engineering." From July 1997 until July 1999, he was an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS PART II, and since January 1998, he has also been an Associate Editor for the IEEE TRANSACTIONS ON NEURAL NETWORKS. He was Chief Guest Editor of the IEEE TRANSACTIONS ON NEURAL NETWORKS 2003 Special Issue on Neural Hardware Implementations.



Antón Civit Balcells received the M.S. degree in physics (electronics) in 1984 from the University of Seville, Seville, Spain. He received the Ph.D. degree in hierarchical multiprocessor design in 1987.

After working for several months with Hewlett-Packard, he joined the University of Seville. In the late 1980s, he participated in the creation of two SMEs related to eLearning and environment monitoring networks. Since 1990, he has been an Associate Professor of computer architecture with the University of Seville. Initially, he worked in research projects related to multiprocessor multiple robot control architectures. He published several papers and directed four Ph.D. degree theses on these topics. As Director of the Robotics and Computer Technology Research Group, he led projects related to advanced wheelchair navigation and intelligent environment support for wheelchair users. These topics have also produced publications and two Ph.D. degree theses. Since 1992, he worked on web and computer accessibility issues. He participated in the creation of the Department of Computer Architecture, University of Seville, where he is currently the Director. He is participating in several EU and national level research projects in the areas of neuromorphic systems, accessible telecoms, and ambient intelligence.

Dr. Balcells is a member of the European Commission eAccessibility expert group. He has been a member of the COST219bis research action management committee and, currently, participates in the COST 219ter (accessibility to next generation networks) management committee. He has participated in several European Commission evaluation activities in the Telematics Applications and IST programs.